Meta-model for Object-Oriented Hierarchical Systems

Lam-Son Lê, Alain Wegmann
Laboratory of Systemic Modeling
Swiss Federal Institute of Technology, Lausanne
EPFL-IC-LAMS, CH-1015 Lausanne, Switzerland
{LamSon.Le, Alain.Wegmann}@epfl.ch

EPFL Technical Report IC/2004/47

Abstract

In enterprise architecture, the goal is to integrate business resources and IT resources in order to improve an enterprise's competitiveness. In an enterprise architecture project, the development team constructs a hierarchical model that represents the enterprise. Our goal is to develop a hierarchical object-oriented model that can serve as enterprise model. The state of the art in hierarchical object-oriented models is such that no models are suited for that purpose. Existing hierarchical models either represent the hierarchy in an inconsistent way, or focus on only one system of interest, or inadequately model actions, or lack a complete meta-model to keep the traceability across the hierarchy. In this paper, we present a meta-model that defines a hierarchical object-oriented model of systems such as those found in enterprise architecture. Miller's organizational levels and the Reference Model of Open Distributed Processing are the foundations of our meta-model that is currently implemented in a web-based CAD tool using Java technology. This paper ends by presenting an example developed with our CAD tool.

Keywords: enterprise architecture, system engineering, Catalysis, hierarchical object-oriented model, meta-model, RM-ODP, Living System Theory.

1. Introduction

The enterprise architecture's goal [4] is to integrate business resources and IT resources in order to improve an enterprise's competitiveness. Enterprise architecture (EA) deals with hierarchical systems that typically span from business entities (market, company, department…) down to IT components (e.g. applications, applets, servlets, bean, COM…). Our goal is the development, in the field of EA, of a method called SEAM [4] and of CAD tools. In this paper, we propose a meta-model that is particularly suitable for modeling hierarchical systems and that is used in our CAD tool.

In EA, multi-disciplinary teams of specialists, under the guidance of an architect, develop an enterprise model stored in and manipulated by CAD tools. To make these CAD tools, we need a meta-model that describes the enterprise model (i.e. the object-oriented representation of hierarchical systems). Today, some meta-models for object-oriented (OO) hierarchical systems already exist. They are defined in the context of design methods that address business and IT such as: Catalysis [2], System Engineering [5], KobrA [7] and Object-Process Methodology (OPM) [10]. Catalysis, defined by D'Souza and Wills, is a development process that addresses business and software. Catalysis analyzes and designs at three levels: business, IT system and software components. Catalysis uses its own modeling notation, inspired by UML [1]. System Engineering (SysEng) defined by the Object Management Group [12] is a development process that addresses the design of systems in general (e.g. airplanes) [5]. First, the developer identifies the stakeholders involved with the system of interest then models the system of interest with multiple levels of subsystems and components. KobrA [7], developed by Atkinson et al., addresses component-based software development. It proposes a recursive model that describes the IT system, its components and the Java classes. Kobra and SysEng use the UML notation. OPM [10], defined by Dori, addresses the modeling of systems in general and can be used from business processes modeling down to software implementation modeling. OPM defines three main model elements: object, process and state, together with some mechanisms that help the modeler define a hierarchy of systems [11]. The notation is specific to OPM.

Unfortunately, the meta-models defined in these methods do not tackle the hierarchy adequately. In this paper, we identify the problems of those meta-models and

propose a new meta-model (used in SEAM) that fits the representation of hierarchical systems.

To give a concrete example, we discuss a bookstore company whose management decides to provide the company's services via Internet. The management creates an EA team who is in charge of the project. In this project, an enterprise model is developed. We use this example to illustrate hierarchical modeling and the related meta-model problems and solutions.

The bookstore enterprise model is made of levels. The *market level* represents business systems, composed of companies, competing to sell books in a market called *BookCoMarket*. The *business system level* represents companies or individuals working together to achieve a commercial goal. In our example, the business system of the on-line book company (*BookCoBis*) is composed of the book publisher (*PubCo*), of the company itself (*BookCo*), of the shipping company (*ShipCo*) and of the bank. The business system of the customer is composed of the customer, the bank and the shipping company that delivers the books. The *company level* represents the departments operating inside the companies. More specifically, in this example, *BookCo* has a purchasing department (*PurchasingDept*) that collaborates with the warehouse department (*WarehouseDept*) for processing the customers' orders. The *department level* represents employees and IT systems. In our example, the purchasing department consists of a clerk and of an order processing application (*OpApp*). One of the main goals of the project is to redesign *OpApp*; but the project also needs to redefine the responsibilities of the employees and of the departments. Note that it would be possible to have additional levels for describing the IT system implementation (e.g. *server level*, *component level* and Java *class level*).

In such a project, the EA team has at least four challenges. First, the team must identify the different systems (or objects[1]) and decide in which hierarchical level they exist (e.g. the market, the business, the company, the department level…). Second, the team has to model in each level the actions happening between the systems belonging to the same level, as well as the corresponding responsibilities and properties of the systems. Concretely, it needs to represent the fact that *BookCo* must interact with *ShipCo* and *PubCo* in order to serve the customer business system. With respect to the action happening between them, *BookCo* has the responsibility to find the books based on the description written in the orders that were issued by the customers. *ShipCo* is responsible for delivering books (perhaps already packed by *BookCo*) and *PubCo* is responsible for providing ordered books that are in initially unavailable in *BookCo*'s warehouse. Objects in other levels such as *BookCoBis*, *PurchasingDept*, *WarehouseDept*… should be represented in the same way. Third, the EA team sometimes has to design and implement more than one system in parallel. For instance, both *WarehouseDept* and *PurchasingDept* need to have new business processes, or the application *OpApp* and the clerk need to have, respectively, new functionalities and job descriptions. Fourth, as there are multiple levels in the model, the team really needs to maintain the traceability between levels: from *BookCoMarket*, to *BookCoBis*, to *BookCo*, to *PurchasingDept* and to *OpApp*. To develop a method and a CAD tool that address these four challenges, a rigorous approach of hierarchical modeling is necessary.

We now give a practical illustration of the problems Catalysis, Kobra, SysEng and OPM might have in representing the on-line bookstore.

- In Catalysis, the modeler can start by describing the *OpApp* application by using Catalysis types. She then can build the business model to describe how *OpApp* interacts with the employees in *PurchasingDept* by means of some joint actions (expressing the goals the developers have about systems). However, Catalysis fails to explain why the book company needs to collaborate with a shipping company that all together makes up a business system. This limits the support of the method when the EA team needs to think about who should be in the business system of *BookCo* to support the new business goal. Moreover, Catalysis cannot solve the problem where both *WarehouseDept* and *PurchasingDept* need to be designed in parallel because it has only one system of interest in the model.
- KobrA is a component-oriented approach that mainly deals with software development. In this method, the modeler is likely to start by specifying one single component for the *OpApp*. The modeler then develops a Functional Model that describes the *OpApp*'s operations and states, which are visible from outside the component. Then a Structural Model and an Interaction Model describe the component realization. These models describe, for example, the interactions between several servlets and an applet. Next, a Functional Model specifies each servlets and the applet. These servlets and the applet are then realized by even smaller components (leading ultimately to Java classes). However, even if the method is recursive for the IT modeling, it fails to represent *ShipCo*, *PubCo* and *BookCoBis*. In addition, KobrA does not represent the so-called Catalysis joint actions between *BookCo*, *PubCo* and *ShipCo*.
- In SysEng, the members of *CustomerBis* are identified as the *BookCo*'s stakeholders (typically

[1]A hierarchical system contains nested systems. As the methods we survey are object-based, the word "object" is frequently used. In this paper, "system" and "object" are synonymous.

shown as UML actors) whereas *BookCo*, *ShipCo* and *PubCo* are represented as systems/subsystems. *BookCo* is part of a larger system called *BookCoBis* and is composed of sub-systems *PurchasingDept* and *WarehouseDept*. The *OpApp* and its implementation units are then likely to be represented as components. The fact that *BookCoBis*, *BookCo* and *OpApp* are different kinds of modeling concepts (i.e. stakeholder, system and component) is inconvenient for the modeler as it makes the hierarchical model inconsistent between levels. In addition, no other actions than *BookCo*'s use-cases are represented in the model. There is no way for the modeler to represent actions happening between *BookCo*, *PubCo* and *ShipCo*, as she can do with so-called joint actions in Catalysis.

- In OPM, the modeler can start by having a physical object called *BookCoMarket*, which is linked via an aggregation-participation to *BookCoBis* and to another physical object representing the customer business system. Similarly, *BookCoBis* has an aggregation-participation link to *BookCo*, *PubCo* and *ShipCo*. A physical process represents the action happening among these three. With the tool OpCat [11], the modeler can zoom-in in *BookCo* to reach *PurchasingDept*, *WarehouseDept* and possibly some processes between them. She can also zoom-in in the *PurchasingDept* and even in *OpApp*. OpCat, by providing the zoom functions, allows the modeler to navigate in the hierarchy. However, the tool does not represent an overall view of the hierarchical levels from *BookCoMarket* down to *OpAp*. The process and meta-model of OPM do not explicitly deal with levels and does not describe the traceability between them.

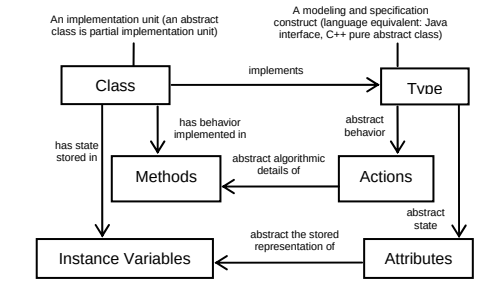
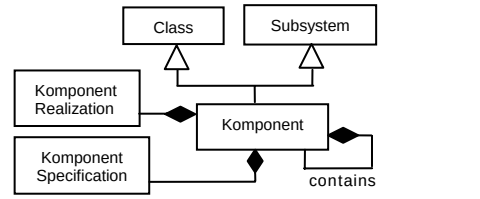
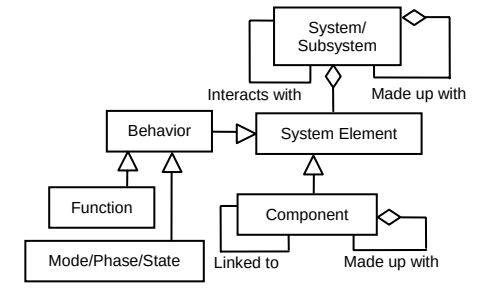
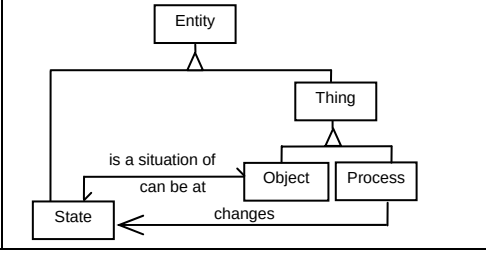
To summarize this analysis, we can conclude that: current methods have difficulties in dealing with models that span through an arbitrary number of levels and have problems representing certain kind of actions happening between systems (e.g. the so-called Catalysis joint actions). With SEAM, we have developed a meta-model that provides the necessary means for the modeler to represent systems/subsystems consistently in all hierarchical levels. Each system is seen either as composite (to reveal, in the next level, which subsystems comprise it) or as whole (to provide a model-based description that conceptually specify them [20]). The action happening between systems, together with the model-based descriptions of these systems, are considered either as whole (usually to express goals) or as composite (commonly to express means to achieve the goals). As a result, our meta-model is constructed to explicitly show how the model elements such as system, actions and state can be organized across the hierarchy.

This paper has the following structure: Section 2 gives a survey of the existing meta-models and points out their limitations. Section 3 defines our meta-model. Section 4 illustrates our meta-model with a concrete example. Section 5 is the conclusion.

2. State of the art and problem definition

In this section, we present how the Catalysis, Kobra, SysEng and OPM's meta-models describe hierarchical systems and the problems they may have in doing this. They are first extracted and then analyzed in terms of their advantages and shortcomings with respect to the 4 challenges introduced in Section 1. At the end of this section, a list of problems summarizes the analysis.

Table 1. Extracted meta-models

Methods	Extracted meta-model
Catalysis [2, 3]	
KobrA [7, 9]	
SysEng [5, 8]	
OPM [10, 11]	

In Catalysis (Table 1), the system of interest (SOI) is initially considered as a type, whose specification consists in a static model (i.e. representing attributes) and an action model (i.e. representing actions). The composition hierarchy is not visible in the meta-model since the meta-model represents one system only. The composition would appear as a hierarchy of types that describe the system of interest, its components, the components of these components, etc…. At the lowest level, when the system needs to be implemented, implementation classes are defined. The hierarchy of types is nevertheless established via different kinds of refinement in an ad-hoc pattern-based manner. (i) However, even if Catalysis is recursive in its approach, it practically deals with 3 levels only (IT system, software components and programming classes). (ii) On the strong side, Catalysis introduces two variants of actions for the action model: the *joint action* (JA) and the *localized action* (LA) [2]. Roughly speaking, a JA models an action happening between several participants (systems, people, organizations…) whereas LAs represent the responsibility of each participant. This action classification is particularly well suited for hierarchical systems development because a LA done by a system may be implemented by a JA that involves its sub-systems interacting; this JA can then be transformed into the relevant LAs for the sub-systems. (iii) Catalysis implicitly assumes that only one system of interest is modeled at a time. This assumption is inadequate in EA because quite often the role of multiple IT systems and employees need to be designed in parallel. (iv) Catalysis Concept Map [3], which could be considered as a meta-model for the method, gives textual and diagrammatic definitions of type, class, JA, LA, attribute… and shows their relations (e.g., a class implements a type, whose abstract behavior is characterized by actions and abstract state is specified by attributes). There are a total of 10 diagrams in this Concept Map. They are difficult to relate and do not describe the concept of level.

In KobrA (Table 1), the systems are considered to have nested components. A component can be either a UML subsystem [1] or a flat class. The two kinds of descriptions of a component (specification and realization) serve as a basis to model the entire system consistently across the hierarchy in a top-down manner. KobrA is naturally suited for hierarchical systems since it defines a recursive top-down approach that allows the modeler to interleave the specification (i.e. the Komponent Specification in Table 1) and the realization (i.e. the Komponent Realization in Table 1) of the components starting from a top-level context [7]. Both the specification and realization appear in various hierarchical levels. (i) However, since KobrA mainly targets software engineering, it does not model the non-IT levels. (ii) Actions happening between components are modeled using UML interaction diagrams with many messages sent between components. This way of representation does not have a single action such as the Catalysis's joint action. (iii) Kobra focuses on one system of interest, the IT system to be developed. However, since Kobra is truly recursive, it could be conceivable to model multiple systems in parallel. (iv) Kobra meta-model is defined as a set of diagrams indicating what UML diagrams should be drawn for the specification, realization and implementation of an individual component [9]. Consequently, the modeler tends to use various UML diagrams. Again, the representations of different components are isolated and thus do not imply the traceability between levels.

In SysEng (Table 1), the Conceptual Model defined in [8] addresses the hierarchical modeling via the loop on the system/subsystem as well as on the component. Both of them have the loop aggregation "Made up with" (see Table 1). The system/subsystems are likely to be used for the upper (or business-related) levels and the components for the lower (software-related) levels. (i) As we can see, SysEng still makes a big difference between the systems and the components, which means that there is a difference at the meta-level between the business levels and the software levels. A truly generic system engineering method should have a more general way to address levels. In our experience, an EA project can have up 10 levels that are split between marketing levels, business process levels and IT levels. Therefore, it would be preferable to have a generic way to deal with all levels. The model elements in each levels can be labeled in a different way to reflect what they represent (e.g. IT system or software component). This name change is an important cosmetic choice as it makes the model more understandable. However, even if the names are different, the way the model elements are dealt with can be the same (so the meta-model can be generic). (ii) In addition, the fact that all actions at the business level are described with use-cases regarding stakeholders has limitations [19]. As such, the modeler has no way to describe actions happening among subsystems or components themselves. This weakness is also shared by the other methodologies that follow the actor-and-use-case approach. (iii) In SysEng, the modeler can design on more than one system/subsystems of interest. (iv) Finally yet importantly, SysEng's meta-model does not explicitly indicate the traceability between modeling levels.

In OPM (Table 1), the first building block is the object. An object can be systemic or environmental (depending if it represents the system of interest or its environment) and can be physical or "informatical" (depending on whether it represents the matter/energy or information). Process is the second building block, which is also categorized as physical or "informatical". A process creates / modifies / destroys one or more objects

in terms of their state – the third building block in OPM meta-model. Objects and processes are connected using links. With a CAD tool called OpCat that was particularly developed for OPM, the modeler can manage the complexity of a hierarchical system that is mainly related to the levels via two mechanisms: in-zooming / out-zooming for exposing or hiding the component objects or component processes and unfolding / folding for exposing or hiding the states of an object [11]. Thanks to a simple but efficient naming convention, the method establishes the equivalence between a textual and a diagrammatic description. (i) The method proposes to use physical objects in non-IT levels and have "informatical" objects and processes model the information the IT system (assumed as the only one system of interest) has about its environment [10]. Therefore, from a standpoint of the level, as the IT system is to the lowest level, the OPM designer no longer has physical objects and processes to model the way the IT system is built. In fact, IT systems have servers, servlets and software components (and even executable code) that can be qualified as physical objects. In contrast, OpCat does accept this kind of modeling and thus allows for modeling hierarchical systems in their whole generality. (ii) By means of physical processes, OPM can represent joint actions such as defined in Catalysis. (iii) Although OPM seems to be system centric, OpCat does allow for representing multiple systems of interest. (iv) Unfortunately, the process and meta-model of OPM do not explicitly describe the traceability along the hierarchy. It is also difficult for the modeler to relate different hierarchical levels with OpCat because the tool does not display any hierarchical views for the entire structure of the model.

Table 2. Problems in modeling hierarchical systems

	Inconsistent levels	Inadequate representation of actions	Only one system of interest	No traceability
Catalysis [2, 3]	x		x	x
KobrA [7, 9]	x	x		x
SysEng [5, 8]	x	x		x
OPM [10, 11]				x

Table 2 summarizes our analysis. It points out which problems each method has in representing hierarchical systems.

i) Inconsistent levels: Just a few levels are taken into account in the model (Catalysis and KobrA). Modeling in different levels is done in various ways (SysEng).
ii) Inadequate representation of actions: What happens between objects is not represented as Catalysis-like joint actions (KobrA and SysEng). In certain levels, actions between objects could even be missing (SysEng).
iii) Only one system of interest: The model is normally built for only one system of interest (Catalysis).
iv) No traceability: due to the lack of a complete meta-model, representations of the same object or action in consecutive levels are not traceable (Catalysis, KobrA, SysEng and OPM).

3. The SEAM meta-model

In this section, we first present the foundations on which our meta-model is built (Section 3.1). We then present our meta-model (Section 3.2).

3.1. Foundations for hierarchical modeling

The foundations of our meta-model come from Miller's level [14] (Section 3.1.1), from our interpretation of the Reference Model of Open Distributed Processing (RM-ODP) [6] (Section 3.1.2) and from the teleological principle (Section 3.1.3). These foundations are briefly presented here.

3.1.1. Miller's level. James Greer Miller introduced the concept of level in [14]. He made a thorough cross-discipline analysis and synthesis of the functions and behavior of living systems. He published his results in 1978 (first edition) and in 1995 (second edition) [14]. His theory is called "General Theory of Living Systems" or "Living Systems Theory" (LST). To develop his theory Miller analyzed 4000 publications from multiple living systems disciplines. He then developed a model that can be used to reason about any living system (from individual cells to supranational organizations such as the United Nations Organization).

The goals of LST are to unify scientific and often discipline-specific approaches in order to study and model livings systems. In particular, LST takes an integral approach to structural and behavioral sciences and still leverages on individual disciplines. On one hand, LST applies the same general theory (i.e., it uses the same concepts and principles) recursively at all levels of living systems. On the other hand, in each level, LST factors in level-specific theories coming, for example, for biology, medicine or social sciences. The result is an original combination of genericity and specificity.

One of the most important concepts is that of *level*. According to Miller "…the universe contains a hierarchy of systems, each more advanced or 'higher' level made of systems of the lower levels". He identifies seven distinct levels for living systems: cells (free-living cells and aggregated cells), organs, organisms (such as humans…), group (such as families, workgroups…), organization (such as commercial companies…), society (such as countries) and supra-national systems (such as inter-governmental organizations …). This level distinction is tightly linked to people's experience in perceiving and studying the world of livings systems. Depending on the goal of the modeler, it is possible to have more or less levels.

To be able to model enterprises, the meta-model should define the model not only confined to IT and software-intensive systems but also to business-related systems. In SEAM, the enterprise model is structured in level that is called *organizational level*. In the example of the on-line book company discussed earlier, there are six organizational levels: market, business system, company, department, IT application and technology level.

3.1.2. RM-ODP. Within the levels, we use RM-ODP to represent what is perceived. RM-ODP is a standard that defines the concepts necessary to build "distributed information processing services to be realized in an environment of heterogeneous IT resources" [6]. RM-ODP also proves to be suitable for general modeling. SEAM meta-model [21] relies on the concepts defined in RM-ODP.

According to RM-ODP part 2 (i.e. the foundations), an entity is any concrete or abstract thing of interest in the universe of discourse. An entity can be considered as atomic or as non-atomic (i.e. composed of parts of the same kind). An entity is represented in the model as a model element. So, a model element can be seen as whole or as composite. A system may be referred to as an entity. A part of a system may itself be a system, in which case it may be called a subsystem. The model element that corresponds to a system is an object. An object can be seen as whole (i.e. this corresponds to the external view of the object, also called model-based specification [20]) or as composite (i.e. this corresponds to the internal view, or implementation, of the object). Other kinds of entities can be modeled as action and state. Action and state can also be seen as whole or as composite. An action (state) seen as composite can be broken down into component actions (states); these component actions (states) can be further broken down into smaller component actions (states). This hierarchy of actions (states) corresponds to the *detail level hierarchy*. The detail level hierarchy includes *detail levels*. It is orthogonal to the organizational level hierarchy defined in the previous section (in which an object is broken down into its component objects). In each organizational level, we can find multiple detail levels. All the terms we use above are formalized in Alloy [16]. This formalization is itself based on Tarski's declarative semantics and on a general system modeling paradigm, developed by Andrey Naumenko, called Triune Continuum Paradigm [17].

According to RM-ODP part 3 (i.e. the architecture), a system specification has five viewpoints: enterprise, information, computational, engineering and technology viewpoint. In the context of hierarchical systems, we model organizational levels (cf. Miller) made of *computational objects* (i.e. an object that represents a system). The computational objects can be specified by either an *information viewpoint* or a *computational viewpoint*. An information viewpoint represents a computational object as whole (or seen from outside). It consists of *information objects* (IO) and *localized actions* (LA). The information objects are like attributes and represent the states of the computational objects. The localized actions model the computational object's responsibility. A computational viewpoint represents a computational object as composite (or seen from inside). It consists of computational objects and actions happening between them that we call *joint actions*. Note that as RM-ODP part 2 just defines the term action, we have to define the localized actions (for the information viewpoint) and the joint action (for the computational viewpoint). The names of these actions are taken from Catalysis [2].

3.1.3. Teleology. One way people give meanings to the behavior of the systems they observe, or they build, is by assigning goals to the systems. This is called *teleology*. The term is defined as "the philosophical doctrine that all of nature, or at least intentional agents, are goal-directed or functionally organized" [18].

In SEAM, we consider that the computational objects represent entities that are goal-directed in the sense that they have some goals when collaborating one with another (or that their designers have goals to be fulfilled by the entities). We consider two kinds of goals: detail-level related goals and organizational-level related goals.

The modeler can decide to make the information viewpoint of the computational object. The information viewpoint shows IOs modified by LAs, which are initially represented as whole. The modeler can decide to make a more detailed information viewpoint with a LA as composite (made of many smaller ones). These two information viewpoints correspond to two detail levels. The first one corresponds to the specification of the goal and the second one to the specification of the means. This corresponds to *goals/means along the detail level*.

The modeler can also decide to make the computational viewpoint of the computational object. This viewpoint shows the component computational

objects that make the computational object of interest. This is another of goals/means relationship that we call *goals/means along the organizational level*. Some people call this a specification/implementation relationship. We do not use these terms because they contain an implicit reference to a development process. In fact, the information and the computational viewpoints are complement to each other. Both viewpoints exist independently of any development process.

We can illustrate those two kinds of goals with our example. The JA happening between *PurchasingDep* and *WarehouseDep* is considered to be the mean for the LA Market done by *BookCo*, which is considered as the goal assigned to *BookCo* when collaborating with *PubCo* and *ShipCo*. The goal of *BookCo* represents what the departments of *BookCo* want to achieve together. This goals/means relationship is along the organizational level. As action *Market* can be divided into *Select* (searching for the books the customers ordered) and *Order* (processing the orders), we have goals/means relationship along the detail level.

3.2. Meta-model definition

Having taken our foundations from Miller [14], RM-ODP [6], and the goals / means modeling from the teleological principle, we now proceed in defining our meta-model. In Section 3.2.1, we define the meta-model for objects seen as whole or as composite. In Section 3.2.2, we define the meta-model for the joint actions, localized actions and information objects seen as whole or as composite. In Section 3.2.3, the previous elements of the meta-model are put all together to make a bi-dimensional meta-model that represent a limited number of organizational and detail levels. Eventually, the meta-model is made general to deal with an infinite number of organizational and detail levels. While presenting the meta-model, we highlight how specific problems that were previously identified are solved by our proposition. This section and the next one use screenshots from our CAD tool.

3.2.1. Organizational level in terms of computational objects as whole / composite. Figure 1 and 2 are screenshots that present two views of the same model. In Figure 1, the computational object *Comm* is broken down into two computational objects *Co1* and *Co2* interacting through a joint action *ja1*. *Co2* is viewed as whole and the localized action *la1* stands for the responsibility of *Co2* in its participation to *ja1*. In Figure 2, *Co2* is seen as composite, it has two component objects *Co21*, *Co22*. They collaborate through *ja2* that is the "implementation" of *la1*. Note that Figure 1 and Figure 2 are two representations of the same entity. They coexist in the CAD tool and they are not considered as being related by a specification / implementation relationship (in which the specification comes first and is then replaced by the implementation).

A computational object seen as whole contains information objects that represent either the fact that a localized action executes (IOs called transaction) or elements of "knowledge" (IOs called concepts) or parameters exchanged with the environment (IOs called parameters). These concepts describe "knowledge" of the computational object about itself or about other computational objects belonging to the same organizational levels. The joint action is described with pre-conditions and post-conditions in terms of the participating computational objects as whole. Once a computational object is seen as composite, it contains component computational objects and joint actions. By changing the view of the computational object from the whole to the composite, the modeler descends to the next organizational level.

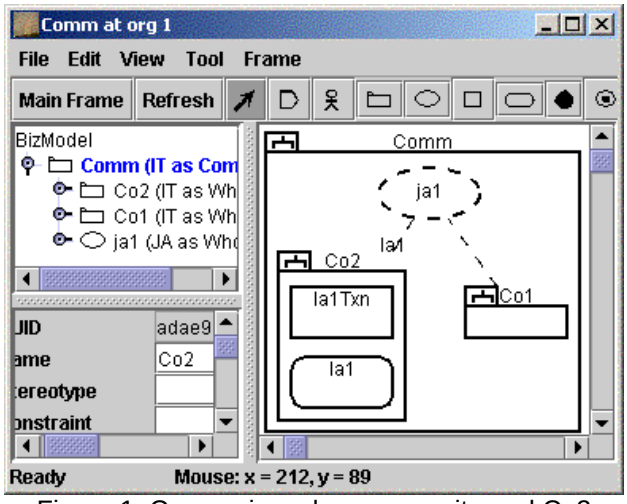


Figure 1. Comm viewed as composite and Co2 viewed as whole

The modeler should not get confused between the LAs that a computational object executes (example *Co2*'s *la1* in Figure 1) and the JAs it mediates (example *Co2*'s *ja2* in Figure 2). The LA *la1* and the JA *ja1* belong to different (and consecutive) organizational levels. Whenever a computational object participates in a JA, its organizational level goals (or the goal its designer has for it) are modeled as LAs. The corresponding organizational level means are represented by the JAs among its component computational objects. In Figure 1, *la1* represents the goal for *Co2*. The means for *Co2* to fulfill *la1* is *ja2* (in Figure 2 below).

The use of organizational levels in which computational objects exist allows the modeler to manage all hierarchical levels (from business down to code) in a systematic way. This computational object can represent

any entities such as markets, companies, IT systems, people, software components, programming classes. Thus, the computational objects can represent stakeholders, the IT systems, the components and even the programming classes.. This illustrates how we solve problem i) "inconsistent organizational level" discussed in the previous section.

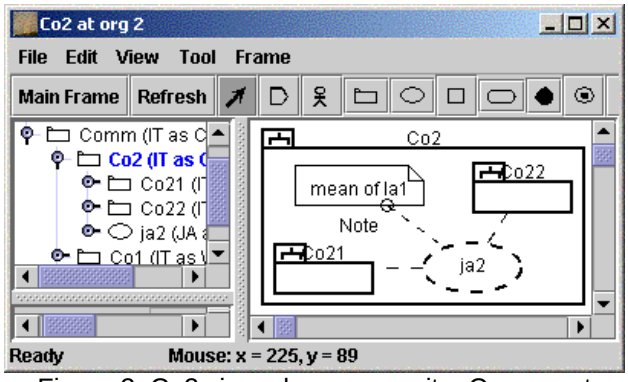


Figure 2. Co2 viewed as composite, Comm not represented.

To be able to build a CAD tool[2] for hierarchical systems [13], we have to choose the notations for those four elements. Actually, in an attempt to make the tool as close to UML as possible, we use graphical elements defined in UML (e.g. UML sub-system for computational objects and UML collaborations for joint actions). The detailed discussion of the notation is out of the scope of this paper.

Let us define the elements of the meta-model with UML class diagrams. Figure 3 describes the meta-model of the computational objects as whole or as composite. They are specialized from a meta-class named Computational Object. In Figure 3 (a), the computational object as composite is broken down into computational objects (seen as whole) that are participants of one[3] JA that is said to be mediated by the context computational object (seen as composite). Note that breaking down a computational object results not only in component objects but also in a *mediator*. The mediator is a computational object that can only be treated as whole. It is responsible for dispatching JAs. For instance, in the example of bookstore, the mediator of the book company corresponds to the way the company exchanges goods and information between its departments (it is an abstract view of the lower organizational levels). In the Java class level, the mediator of a Java class passes parameters to some component class on calling its specific method and typically prepares parameters based on returned values for further invocation. It is an abstract view of the virtual machine.

Figure 3 (b) is the meta-model of a computational object seen as whole. It has one IO to model the entire transaction of a computational object. This IO also represents the whole state of the computational object, which also has one LA to represent its entire behavior (i.e. the lifecycle of the object). This LA probably changes the value of IO and thus affects the state of the computational object. The introduction of JA in every computational object (as composite) together with an explicit LA for each participating object (as whole) overcomes the problem ii) "inadequate representation of object interaction" mentioned in section 2. The problem iii) "only one system of interest" is solved by having any participating objects specified as whole or as composite if the modeler wishes to do so.

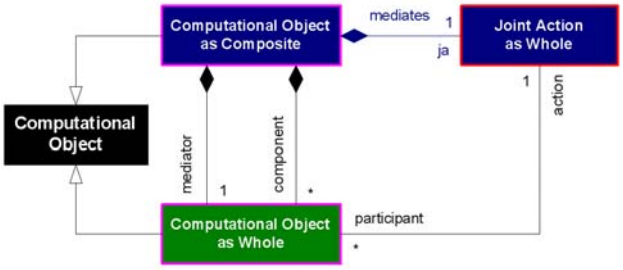


Figure 3: (a) meta-model of computational object viewed as composite

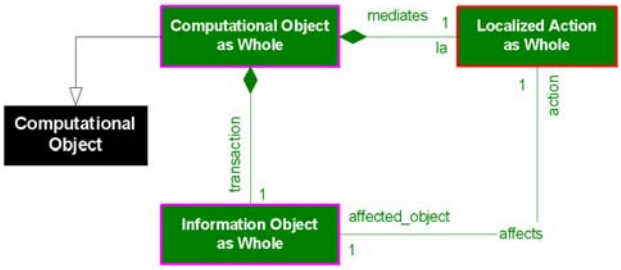


Figure 3: (b) meta-model of computational object seen as whole

3.2.2. Detail level in terms of joint actions, information objects and localized actions as whole / composite. Not only computational objects but also JAs, LAs and IOs can be treated as whole and as composite. Once the modeler breaks down a JA, she descends to the subsequent detail level. Figure 4 is again a screenshot of our CAD tool with the simple example, but now at second detail level. The JA *ja1* (seen as composite) is divided into *ja11* and *ja12*. Correspondingly, the LA *la1* as composite consists of *la11* and *la12* and the IO *la1Txn* as composite is broken

[2]We call it CAD tool as opposed to CASE tool because it also deals with some levels above software engineering.

[3]In the next section, we will present how multiple JA can be represented. They correspond to an additional detail level.

down into *la1TxnSelf*, *la1Txn*, *la2Txn* and *Co1Concept* (knowledge of *Co2* about *Co1*). *la1TxnSelf* happens to be the mediator of *la1Txn* after it is broken down into *la1Txn* and *la2Txn*. The correspondence between the JA, IO, LA as whole and JA, IO, LA as composite that we just described is what we call a goals/means relationship along the detail level. Note that our tool keeps the traceability between the concepts and what it represents in the system's environment. This is done by a "trace" relationship such as the relation going from *Col1Concept* to *Co1* in Figure 4.

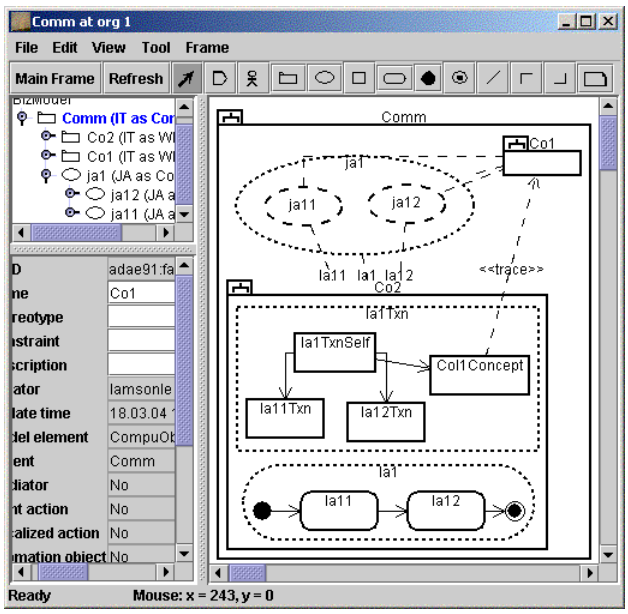


Figure 4. Second detail level of Co2 and of the relevant part of its environment

In Figure 5, we present the meta-model elements concerning the detail levels: breaking down a composite element results in component elements together with a constraint on how they are put together. The explicit constraints for the JA and LA seen as composite could be behavioral constraints (not shown in the Figures 3 and 5). They determine the sequence or the loop of the component actions. The constraint for a composite IO is the relationships between component objects. Note that, just similar to the case of the composite computational object, there is always a mediator for each composite LA and JA. Note also that a meta-class named Joint Action is the generalization of JAs seen as whole or composite. Information Object and Localized Action are the meta-class for the generalization of IOs seen as whole/composite and LAs viewed as whole/composite, respectively.

In general, an IO may represent a transaction (standing for the execution of a LA), a concept (which is the knowledge of a computational object about another) or a parameter. Typically, transactions need to be treated as composite. As indicated in Figure 5 (c), a transaction can be broken down into several sub-transactions, some concepts, a number of parameters and a mediator called "self". According to either Figure 5 (b), a LA as composite can be broken down into some interaction and internal LAs. An interaction LA requires some parameters exchanged between the environment and the computational object that mediates it. In contrast, an internal LA does not require any parameters.

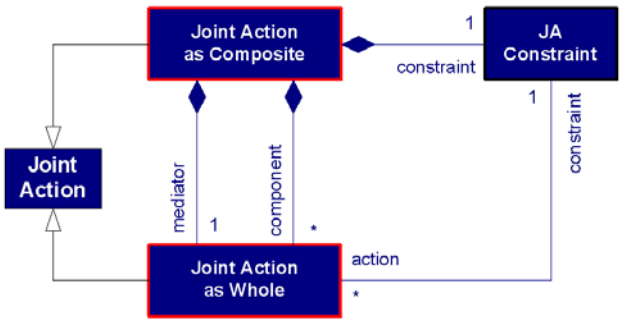


Figure 5: (a) meta-model of JA as composite

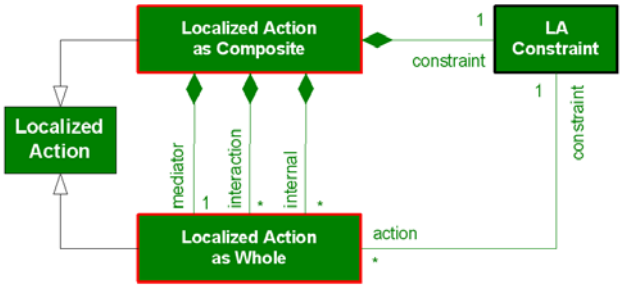


Figure 5: (b) meta-model of LA as composite

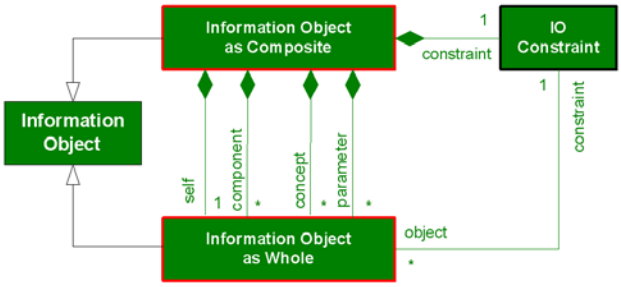


Figure 5: (c) meta-model of IO as composite

3.2.3. Combining organizational and detail levels. We eventually come to a complete meta-model where each element can be toggled between the whole and the composite view (see Figure 6). The postfix of the class name and the tagged value in { } indicate to which organizational and detail level the concept belongs (_x_y means org level x, detail level y). Intuitively, we could

consider that the organizational levels make the meta-model spread vertically and the detail levels horizontally. The meta-classes in Figure 6 are more specific than that in Figure 3 and 5 because they are all specialized with the organizational and detail level.

Once the modeler goes from the whole view to the composite view of a computational object, she changes to the subsequent organizational level. In general, a computational object as composite consists of several computational objects as whole together with a JA at detail level 1. At this level, the component object as whole has a LA and an IO indicating a transaction of that action. The modeler goes into the next detail level if she follows the association from the single JA as whole to it as composite where it is broken down synchronously with the corresponding LAs and IOs of the participating computational objects. For each computational object, the LAs probably change the value of its IOs and thus affect the state of the object. By looking at Figure 6 in terms of columns that are visually formed by the detail level, we can see that the patterns described in Figure 3 are used in the left column whereas the patterns depicted in Figure 5 appear in the middle and the right column.

As discussed in section 2, a complete meta-model is necessary to maintain traceability. So our meta-model addresses problem iv. Thanks to this kind of traceability, a CAD tool can help the modeler navigate those diagrams although they are separately displayed. Such a tool should manage all model elements in a way that diagrams are just extracted view based on some criteria (e.g. some objects, actions as whole or as composite) [13].

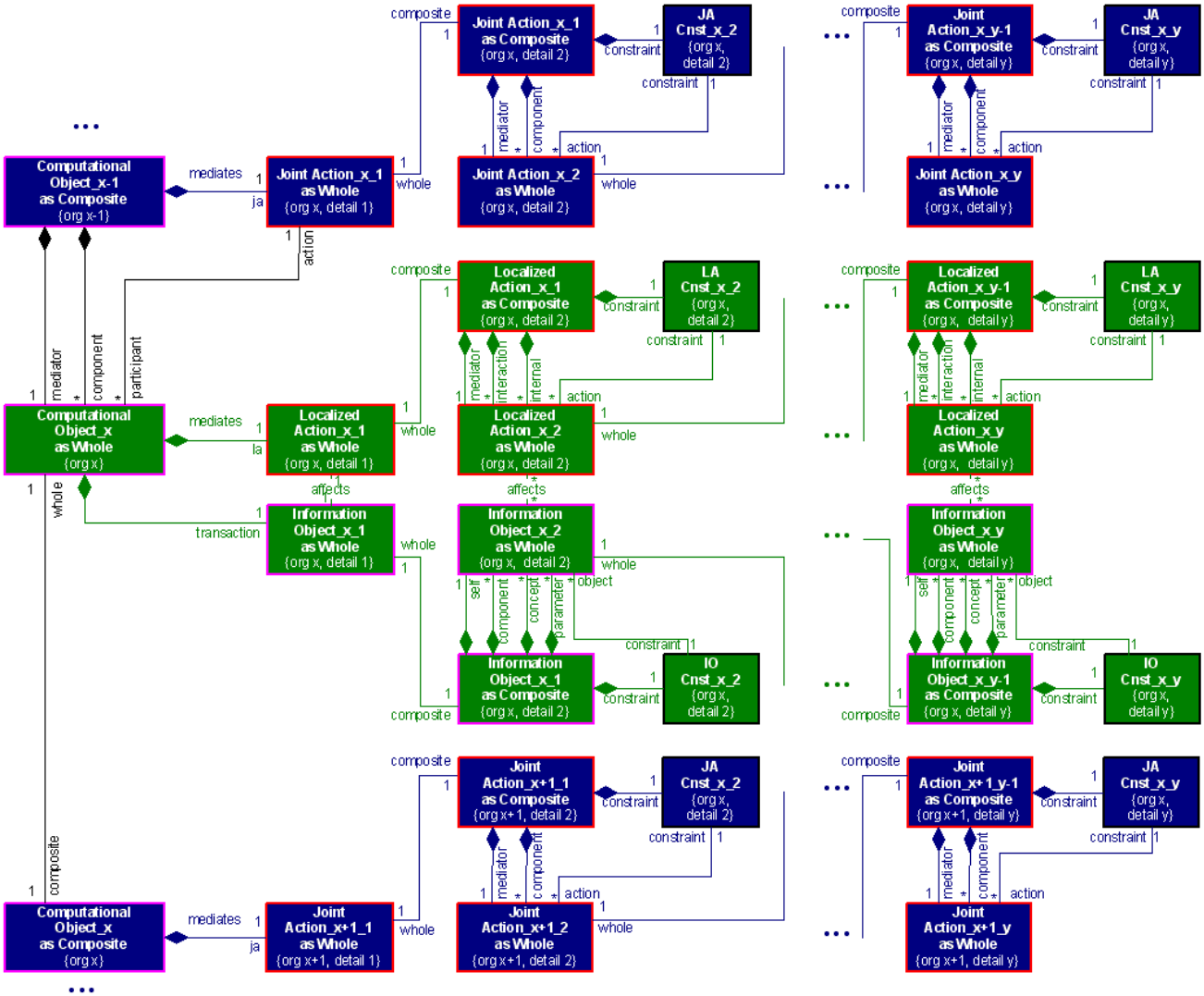


Figure 6. Complete meta-model showing the occurrences of the organizational and detail level

To make the meta-model more general, we group the occurrences of elements of the same kind that appear in each organizational and detail level. The detail level can be inferred via the composition relationship connecting Joint Action, Information Object and Localized Action as composite to the respective generalized element. The constraints of the two roles of this composition say that the element as composite is one detail level above the generalized element. Similarly, the organizational level can be inferred via the composition connecting Computational Object as composite to the generalized Computational Object, which is one organizational level below the composite one.

We eventually come to the simplified but general meta-model as shown in Figure 7 below. Note that in this meta-model, unlike Figure 3 and 5, the targets of the composition relationships should be the generalized meta-classes, namely Computational Object, Joint Action, Information Object and Localized Action rather than their specializations that are viewed as whole. This modification is mainly because the general meta-model copes with infinite organizational and detail levels. The generalized meta-classes are necessary to be able to switch between an element as whole and itself but viewed as composite (see the associations having the role name "whole" and "composite" in Figure 6). The general meta-model depicted in Figure 7 should be compared to the ones shown in Table 1. It makes the organizational level and detail level hierarchies explicit. It also makes explicit the fact that the all entities could be modeled as whole or as composite. This model is a solid base for hierarchical system modeling.

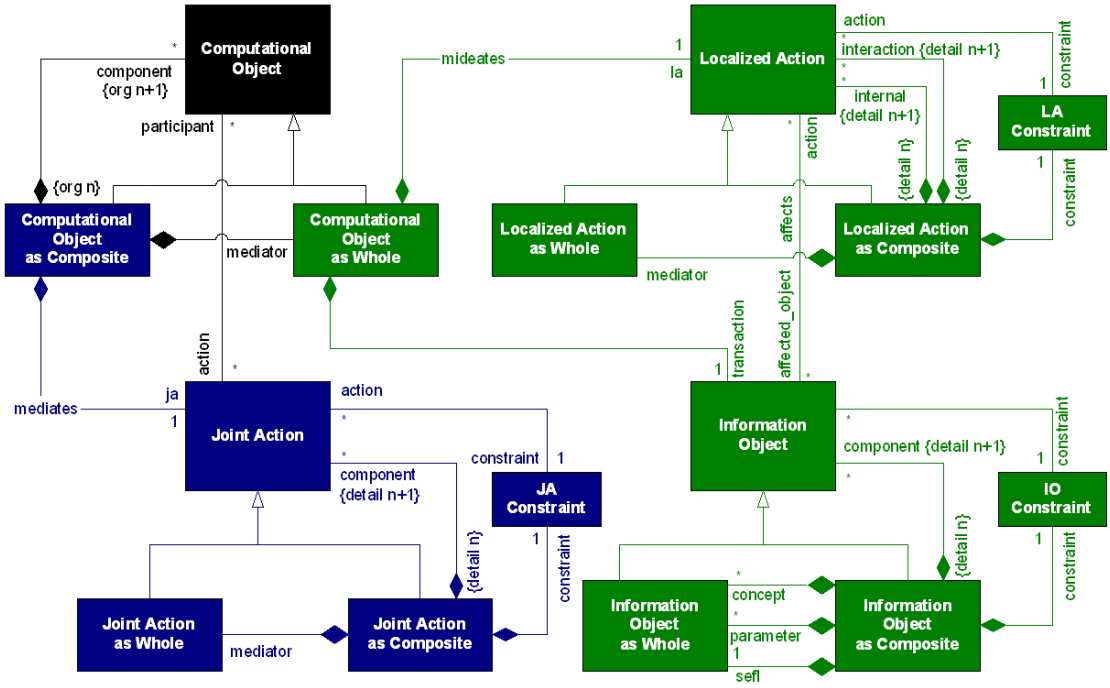


Figure 7. General meta-model

4. Example

In this section, we illustrate our meta-model with the bookstore example given in the introduction. The solution is a series of SEAM diagrams that represent the entire system from the market level down to the binary level. This representation supports the EA team and helps them to address all challenges, as we have defined in Section 1. The example is illustrated with diagrams created from our CAD tool [13] that were redrawn to save space.

In these diagrams, the levels to which the objects and actions belong are visually illustrated with a tagged value {org x, detail y}. As we can see in Figure 8, at the top organizational level, *BookCoMarket* as composite consists of two business systems: *BookCoBis* and *CustomerBis*. The former sales books to the latter based on the description written in orders issued by the members of *CustomerBis*. They collaborate through the JA *sale*. *BookCoBis* is responsible for performing the localized action *Mfg_Sale.*

In the second organizational level, *BookCoBis* as composite is composed of the company *BookCo* that collaborates with *PubCo* and *ShipCo* to fulfill the JA *Mfg_Sale* that is in fact the implementation of the localized action *Mfg_Sale*. This corresponds to the organizational goals/means relationship.

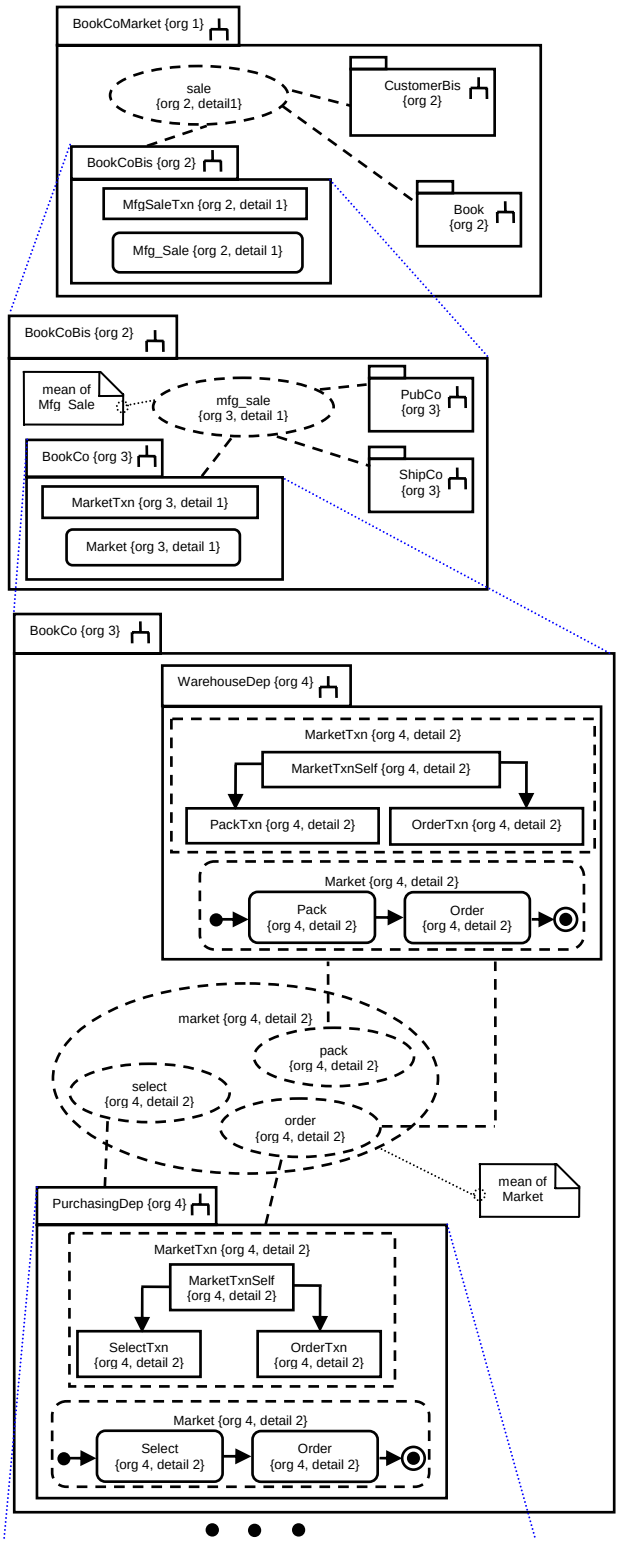


Figure 8. Model of the online bookstore for business-related organizational levels

In the third organizational level, *BookCo* as composite contains two departments: *PurchasingDep* (for selecting books and filling orders) and *WarehouseDep* (for packing selected book). Note that both of them are specified as whole when participating in JA *market*. This representation is actually the mean for LA *Market* of *BookCo* in the previous organizational level. JA *market* as composite has constituent actions *select*, *pack* and *order* that all belong to the second detail level (whereas *market* as whole belongs to detail level 1). Accordingly, *PurchasingDep* as whole is also specified in the second detail level to have *MarketTxnSelf*, *SelectTxn*, *OrderTxn* (IOs representing the transactions) and LA *Select*, *Order*. *WarehouseDep* is different with the appearance of action *Pack* instead of action *Select*. As drawn in the following diagram, both *PurchasingDep* and *WarehouseDep* are specified as whole in the same UML diagram. This practically illustrates the solution to problem iii) presented in Section 3.

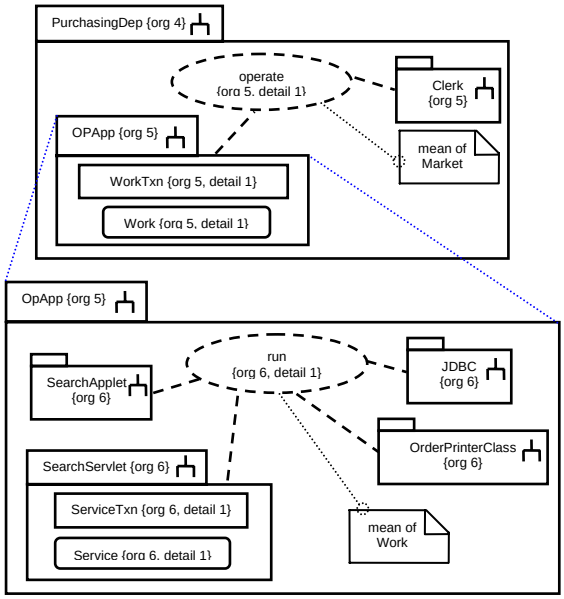


Figure 9. Model of the online bookstore for IT-related organizational levels

In the fourth organizational level, *PurchasingDep* as composite shows that a *clerk* operates a program *OpApp* to search for ordered books and to process received orders. The search result (identification of found books) is then given to department *Warehouse* to locate and pack the actual books. The interaction between the *clerk* and *OpApp* is modeled as a JA *operate*. The goal of *OpApp* is represented as LA *Work*. If the EA team models the technology level, the mean for *Work* is a JA that happens within *OpApp*. We can see that JAs are expressed in every composite computational object. For each JA, the participants have a LA expressing their goal. This

illustrates the solution to problem ii) presented in the previous section.

It is possible to have a fifth organizational level (technology level) where *OpApp* as composite is elaborated to contain *SearchApplet* (a Java applet that interacts with the clerk), *SearchServlet* (a Java servlet responsible for searching) that uses a Java class *OrderPrinterClass* (for filling orders) and calls JDBC (see Figure 9 above). The model we built, so far, has a total of 6 organizational levels. (*SearchServlet* and *SearchApplet* belong to the sixth organizational level, as indicated in Figure 9). Each computational object as composite, which is drawn under a separate UML subsystem notation, is visually traced back by dash lines to the notation representing it as whole within another computational object. With our CAD tool, the modeler just follows the popup menu of each computational object and then can toggle the view or make it as a context object to change to the subsequent organizational level. The tool also displays a tree view for all computational objects and joint actions of the model on the left of each frame (see Figure 1, 2 and 4). This illustrates how we solve problem i) and iv) as presented in Section 3.

5. Conclusion

Existing methodologies and their meta-models have some deficiencies in modeling hierarchical systems: partial hierarchy description, missing joint action, only one system of interest in the model and multiple untraceable representations. To overcome these problems, we propose a complete meta-model that is based on Miller's theory of Living Systems [14], RM-ODP [6] and teleology. The main contribution of this meta-model is the definition of four basic modeling concepts (computational object, joint action, information object and localized action) together with the application of two complimentary views (composite/whole). This allows us to explain the organizational level and define the detail level.

Our CAD tool has been implemented to show how the meta-model practically works. Through various examples, we validated the tool and the meta-model. Our experience with the tool shows that the user interface should be designed with great care. As the models are complex, the tool should reduce this complexity for the user. This is the direction of our future work.

6. References

[1] OMG, UML 1.5 Specification, 2003, http://www.omg.org/technology/documents/formal/uml.htm

[2] Desmond Francis D'souza, Alan Cameron Wills, Object, Components and Frameworks with UML, the Catalysis method, 1999, Addison-Wesley

[3] Catalysis Concept Map, http://www.catalysis.org/overview/concepts/concept-map/graphical-concept-map.htm

[4] Alain Wegmann, On the Systemic Enterprise Architecture Methodology (SEAM), Conference on Enterprise Information System 2003 (ICEIS 2003), Angers, France

[5] System Engineering OMG, http://syseng.omg.org

[6] ISO/IEC 10746-1, 2, 3, 4 | ITU-T Recommendation, X.901, X.902, X.903, X.904, Reference Model of Open Distributed Processing, OMG, 1995-1996

[7] Atkinson, C.; Paech, B.; Reinhold, J.; Sander, T.; Developing and applying component-based model-driven architectures in KobrA, fifth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2001), September 2001, Seattle, USA

[8] System Engineering Conceptual Model (Work in Process), http://syseng.omg.org

[9] Meta-model for KobrA method, http://www.iese.fhg.de/Publications/book/Guides/Metamodel.pdf

[10] Dov Dori, Object-Process Methodology, A Holistic Systems Paradigm, Springer Verlag, 2002

[11] Dov Dori, Iris Reinhartz-Beger, Arnon Sturm, OPCAT – A Bimodal CASE Tool for Object-Process Based System Development, accepted for the fifth International Conference on Enterprise Information Systems (ICEIS), 2003

[12] Object Management Group, http://www.omg.org/

[13] CAD tool for Systemic Enterprise Architecture Modeling, http://lamspeople.epfl.ch/le/SEAMtool

[14] Miller.J.G., Living Systems, University of Colorado Press, 1995

[15] Alain Wegmann, Otto Preiss, MDA in Enterprise Architecture? The Living SystemTherory to the Rescue…, seventh IEEE International Enterprise Distributed Object Computing Conference (EDOC 2003), September 2003, Brisbane, Australia

[16] A. Naumenko, A. Wegmann, G. Genilloud, W.F. Frank. Proposal for a formal foundation of RM-ODP concepts, Proceedings of ICEIS 2001, Workshop On Open Distributed Processing – WOODPECKER'2001, Setúbal, Portugal, July 2001, pp. 81-97

[17] Andrey Naumenko, Triune Continuum Paradigm: a paradigm for General System Modeling and its applications for UML and RM-ODP, PhD thesis 2581, EPFL, June 2002. http://www.triunecontinuum.org

[18] Robert Audi, The Cambridge Dictionary of Philosophy, Second Edition, Cambridge University Press, 1999

[19] A. Wegmann, G. Genilloud, The Role of "Roles" in Use CaseDiagrams, Proceedings of 3rd International Conference on the Unified Modeling Language (UML2000), York, UK, October 2000

[20] Bernhard Schätz, Alexander Pretschner, Franz Huber and Jan Philipps. Model-based development of embedded systems. In *Advances in Object-Oriented Information Systems*, Lecture Notes in Computer Science 2426, pages 298-311, 2002

[21] Alain Wegmann, Andrey Naumenko, Conceptual Modeling of Complex Systems using an RM-ODP based Ontology, fifth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2001), September 2001, Seattle, USA